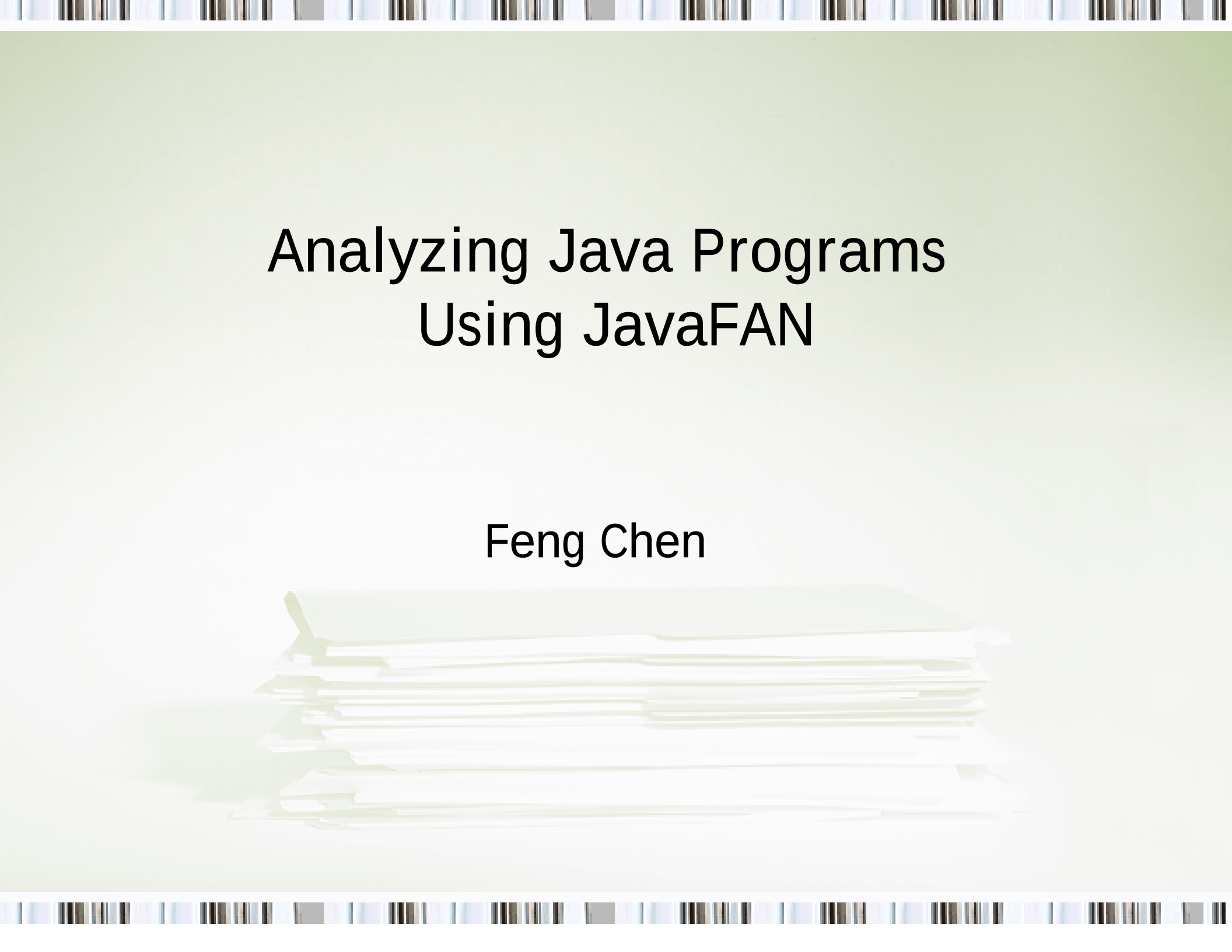
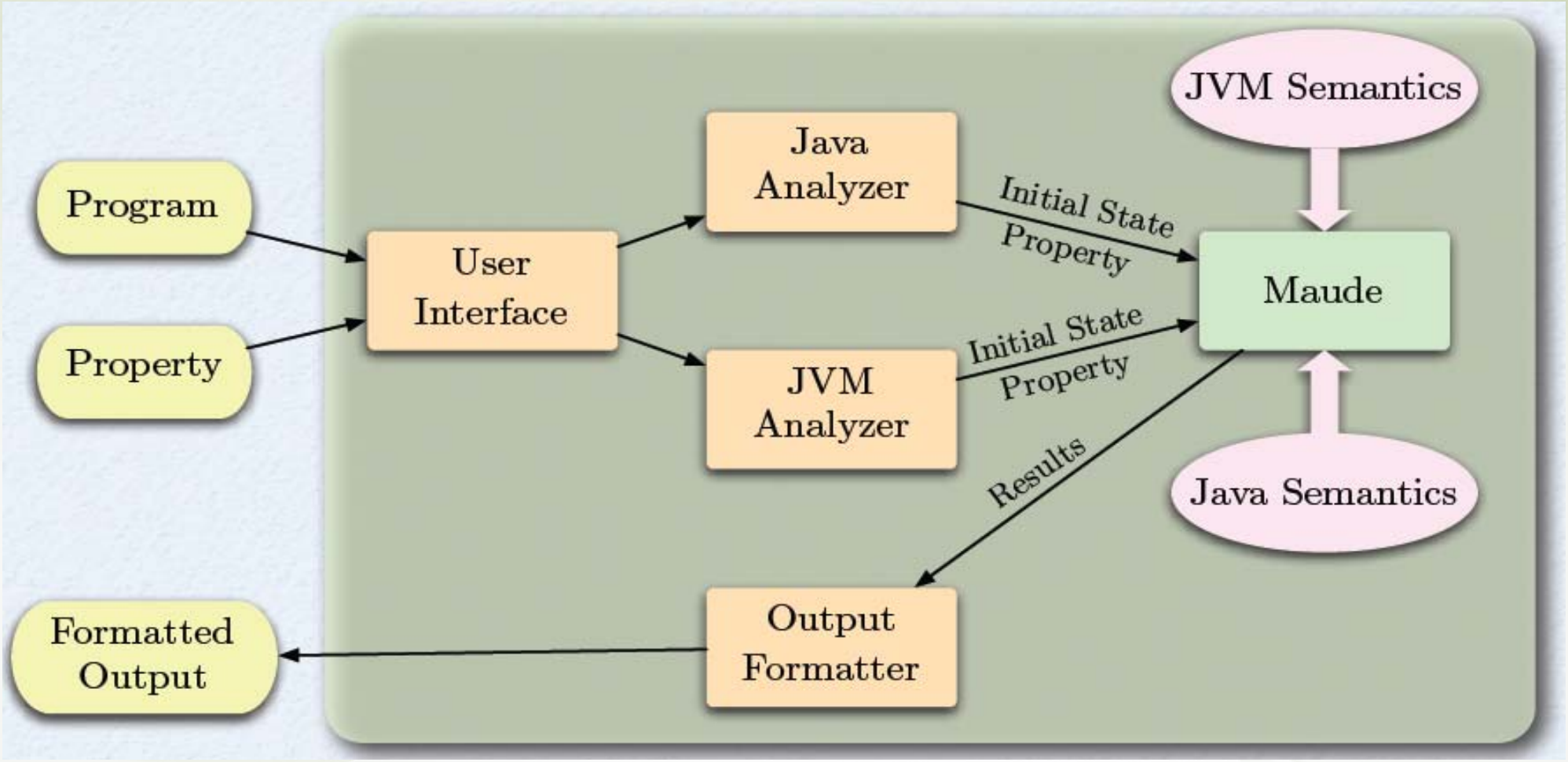
Analyzing Java Programs Using JavaFAN

Feng Chen



JavaFAN (Java Formal Analyzer)



Resource

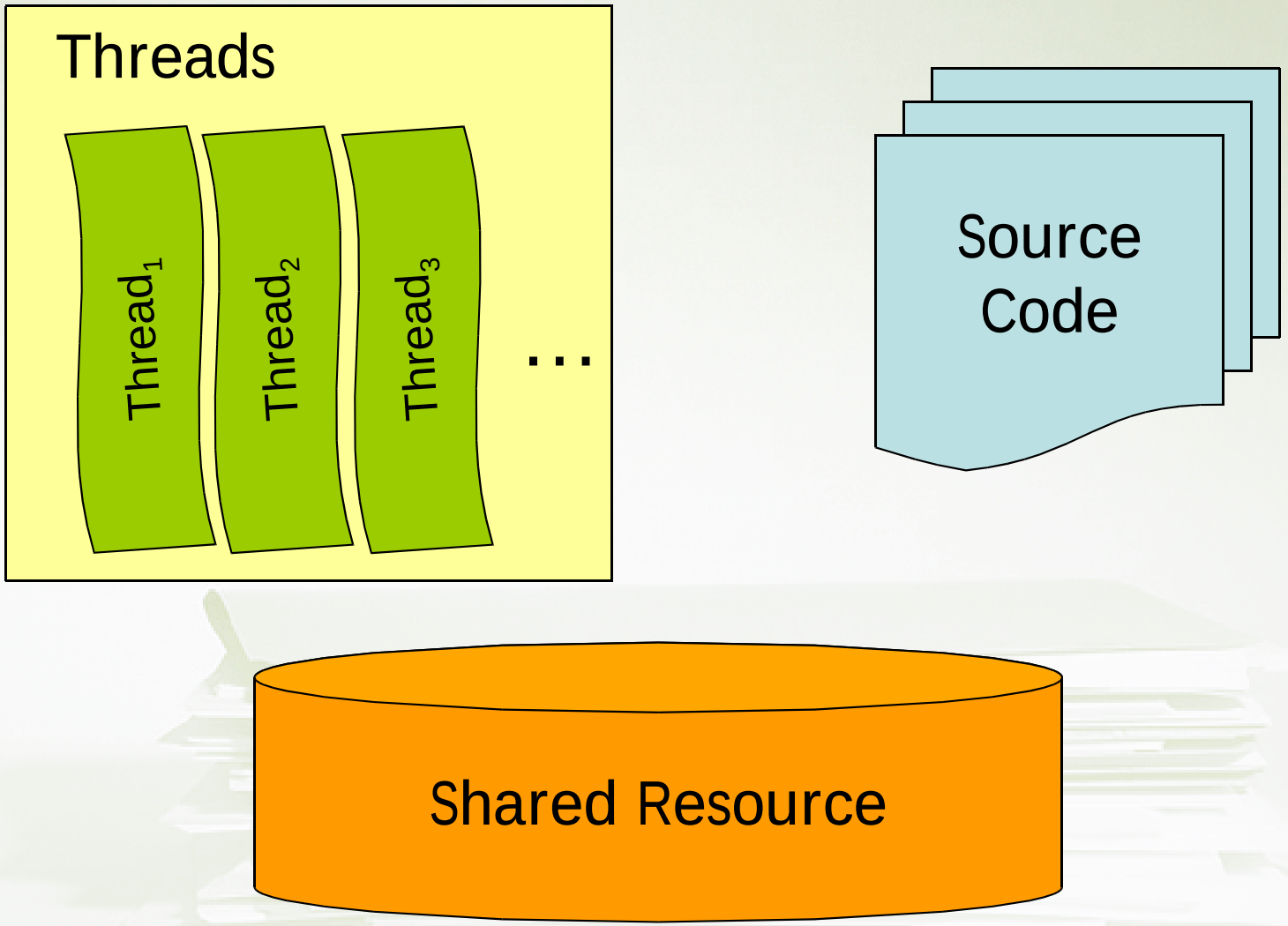
- Official website: http://javafan.cs.uiuc.edu

Inside Java Semantics

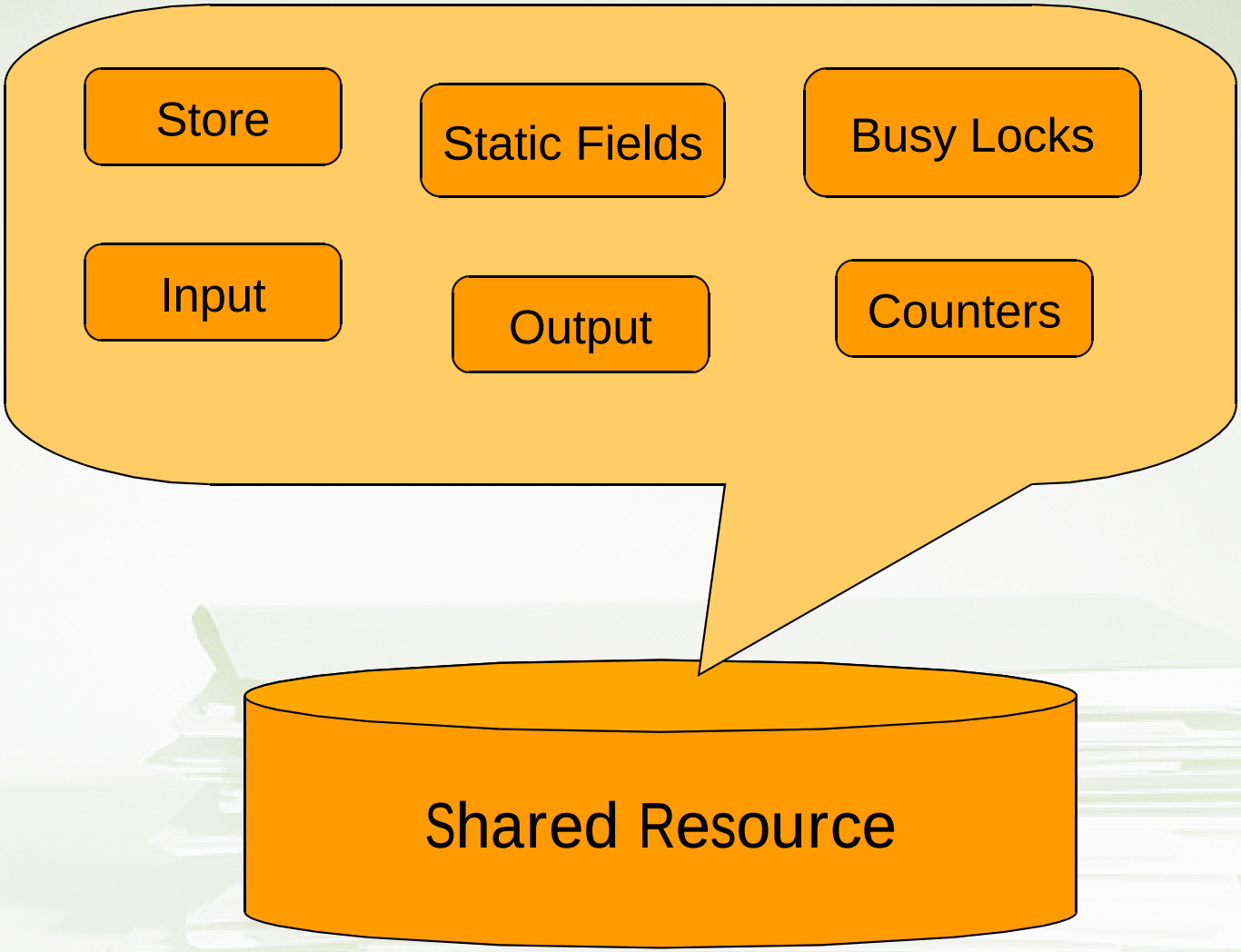
JavaRL (Java Rewrite Logic) Semantics

- An operational semantics
- An execution = a sequence of state transitions
  - State transitions
    - Equations: deterministic sequential state transitions
    - Rewrite rules: nondeterministic concurrent state transitions
  - State definition

JavaState


Threads
Thread1
Thread2
Thread3
...
Source Code
Shared Resource

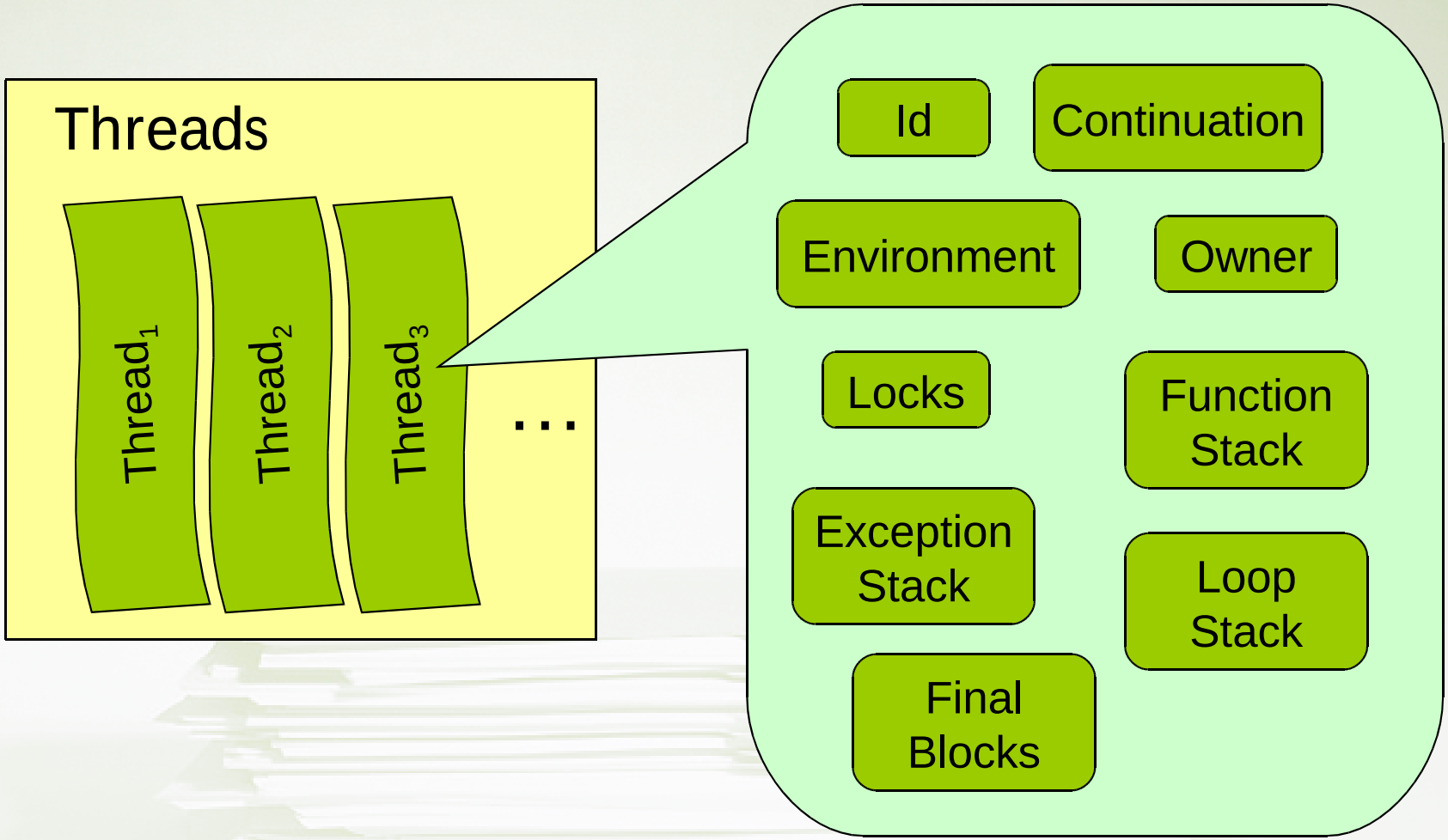
JavaState – Shared Resource



JavaState - Definition

```
sort JavaState .

op code : Classes -> JavaState . --- source code
op t : ThreadCtrl -> JavaState . --- thread contexts
op out : Output -> JavaState . --- output
op in : ValueList -> JavaState . --- input
op store : Store -> JavaState .  --- memory
op static : ObjEnv -> JavaState . --- the static fields of classes
op busy : ObjectSet -> JavaState . --- the locks are in use
op nextLoc : Int -> JavaState . --- the next available location
op nextTid : Int -> JavaState . --- the next thread id
op __ : JavaState JavaState -> JavaState [comm assoc] .
```

ThreadCtrl


Threads
Thread1
Thread2
Thread3
...
Id
Continuation
Environment
Owner
Locks
Function Stack
Exception Stack
Loop Stack
Final Blocks

FlowStack – Method Return

```
eq k(return E ; -> K) = k(E -> return -> K) .

eq t(k(V -> return -> K) fstack(fsi(K', (holds(Ll) env(Env) tc)) fstack) holds(Ll') env(Env') tc')
    = t(k(releaseEnv(Env') -> release(Ll, Ll') -> (V -> K')) fstack(fstack) holds(Ll) env(Env) tc) .
```

ThreadCtrl - Definition

```
sorts ThreadCtrl FlowStackItem FlowStack .
 op id : Int -> ThreadCtrl . --- the thread id
 op k : Continuation -> ThreadCtrl .
 op obj : Object -> ThreadCtrl .
 op fstack : FlowStack -> ThreadCtrl . --- function stack
 op xstack : FlowStack -> ThreadCtrl . --- exception stack
 op lstack : FlowStack -> ThreadCtrl . --- loop stack
 op finalblocks : FlowStack -> ThreadCtrl . --- final blocks
 op env : Env -> ThreadCtrl .
 op holds : LockSet -> ThreadCtrl . --- the locks held by the thread
 op __ : ThreadCtrl ThreadCtrl -> ThreadCtrl [comm assoc] .

 subsort FlowStackItem < FlowStack .
 op noItem : -> FlowStackItem .
 op fsi : Continuation ThreadCtrl -> FlowStackItem .
 op __ : FlowStack FlowStack -> FlowStack [assoc id: noItem] .
```

Environment and Store

Environment: mapping from names to locations

```
sort Env .
op noEnv : -> Env .
op [_,_] : Qid Location -> Env .
op __ : Env Env -> Env [assoc comm id: noEnv] .
```

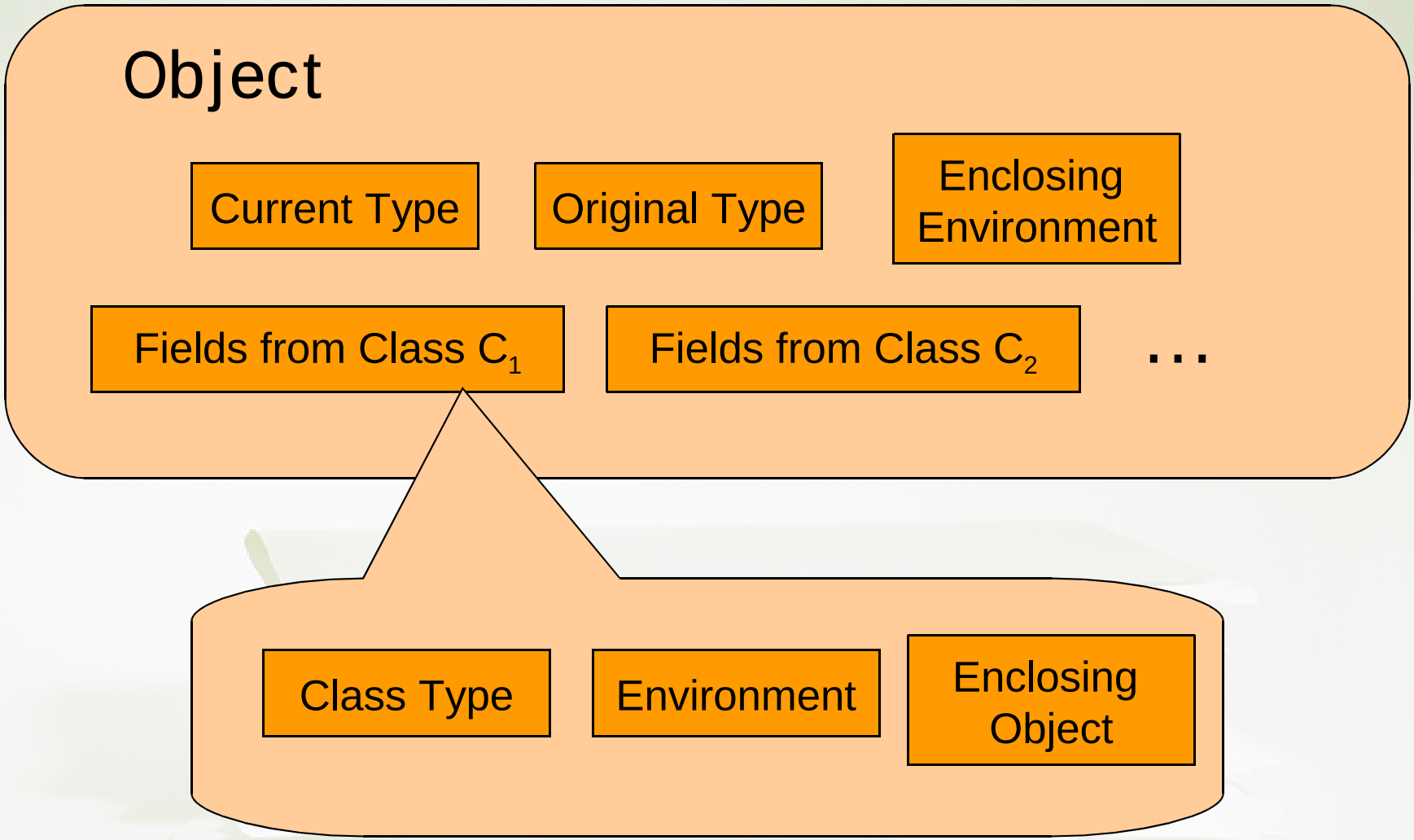
Store: mapping from locations to values

```
sort Store .
op noStore : -> Store .
--- the last bit is used to store the thread id
--- needed for fine grained concurrency and reclaiming locations
op [_,_,_] : Location Value Int -> Store .
op __ : Store Store -> Store [assoc comm id: noStore] .
```

Example: Store Accesses

```
eq t(k(#(L) -> K) id(I) tc) store([L, V, -2] st) = t(k(V -> K) id(I) tc) store([L, V, -2] st) .

rl t(k(#(L) -> K) id(I) tc) store([L, V, -1] st) => t(k(V -> K) id(I) tc) store([L, V, -1] st) .

eq t(k(#(L) -> K) id(I) tc) store([L, V, I] st) = t(k(V -> K) id(I) tc) store([L, V, I] st) .

eq t(k([V -> L] -> K) id(I) tc) store([L, V', -2] st) = t(k(K) id(I) tc) store([L, V, -2] st) .

rl t(k([V -> L] -> K) id(I) tc) store([L, V', -1] st) => t(k(K) id(I) tc) store([L, shared(V), -1] st) .

eq t(k([V -> L] -> K) id(I) tc) store([L, V', I] st) = t(k(K) id(I) tc) store([L, setTid(V, I), I] st) .
```

Object


Object
Current Type
Original Type
Enclosing Environment
Fields from Class C_1
Fields from Class C_2
...
Class Type
Environment
Enclosing Object

Object - Definition

```
sorts ObjEnvItem ObjEnv Object ObjectAttribute .
subsort Object < Value .

op o : ObjectAttribute -> Object .
op __ : ObjectAttribute ObjectAttribute -> ObjectAttribute [comm assoc id: noAttr] .
op noAttr : -> ObjectAttribute .
op f : ObjEnv -> ObjectAttribute .
op curr : Type -> ObjectAttribute .
op orig : Type -> ObjectAttribute .
op encEnv : Env -> ObjectAttribute . --- the env of the enclosing block.
op __ : ObjEnv ObjEnv -> ObjEnv [comm assoc id: onil] .
op onil : -> ObjEnv .
op [_] : ObjEnvItem -> ObjEnv .
op t : Type -> ObjEnvItem .
op enc : Object -> ObjEnvItem . --- the enclosing object
op f : Env -> ObjEnvItem .
op inil : -> ObjEnvItem .
op _,_ : ObjEnvItem ObjEnvItem -> ObjEnvItem [comm assoc id: inil] .
```

Access Object Fields

```
--- the owise option allows the system object take effect first, like system . out
eq k((E . X) -> K) = k(E -> (.(X) -> K)) [owise] .

eq k(o(oattr curr(T)) -> (.(X) -> K)) = k(o(oattr curr(T)) -> (.(X, T) -> K)) .

eq k(o(oattr curr(T) f([oeitem, t(T), f(([X, L] env))] oEnv)) -> (.(X, T') -> K)) = k(#(L) -> K) .

eq t(k(o(oattr curr(T)) -> (.(X, T') -> K)) tc) static([t(T), f([X, L] env)] oEnv) =
    t(k(#(L) -> K) tc) static([t(T), f([X, L] env)] oEnv) .

eq t(k(o(curr(T) oattr) -> (.(X, T') -> K)) tc) code(Cl) =
    t(k(o(curr(getSuper(T, Cl)) oattr) -> (.(X, T') -> K)) tc) code(Cl) [owise] .

eq k(o(curr(Object) encEnv([X, L] env) oattr) -> (.(X, T) -> K)) = k(#(L) -> K) .

eq k(o(oattr curr(Object) f([enc(obj), t(T), oeitem] oEnv)) -> (.(X, T) -> K)) =
    k(obj -> (.(X) -> K)) [owise] .
```

Optimizations

- Minimizing the number of rewrite rules
- Tracking potential shared locations
- Reclaiming store locations
- Partial Order Reduction

Demos

Java Analyzer

java *javarl.JavaRL –cp <path> [options] [MainClass]*

Options: *[-maudecode] [–op <path>] [-mc <path>] [-s deadlock]*

- *-cp <path>*: designate the class path, can be a directory or a Java file
- *-maudecode*: generate the Maude module only
- *-op <path>*: write the output to the specified file
- *-mc <path>*: model check the program against the property specified in the property file
- *-s deadlock*: search for deadlocks
- *MainClass*: the main class in the program; if ignored, the first class with the main function will be used

Interpretation

java *javarl.JavaRL –cp <path> [MainClass]*

```
class Sum {
        public static void main(String[] args)
        {
          int sum = 0;
          for (int i=0; i<10000; i++) sum += i;
          System.out.println(sum);
        }
}
```

Search

java *javarl.JavaRL –cp <path> -s deadlock [MainClass]*

Model Checking

java *javarl.JavaRL –cp <path> -mc <path> [MainClass]*

Property Specification

Syntax:

```
Property    ::= Atom* Formula
Atom        ::= "atom" <name> ":" <className> ("." | "@")
                BooleanExp (“/\" BooleanExp)*
BooleanExp ::= <field> (">" | ">=" | "<" | "<=" | "==") <integer>
Formula     ::= "formula" ":" <LTL formula>
```

Example:

```
atom readers: data@rnum > 0
atom writers: data@wnum > 0
formula : [](~ (readers /\ writers))
```

Property in Maude

java *javarl.JavaRL –cp <path> -maudecode –op <path>*

```
op readers  : -> Prop .
ceq run(store([L, o(objAttr f([t(t('data)), f([ 'rnum, L0 ] env)] oEnv)), I]
      [L0, int(I0), I0'] store) state) |= readers = true if (I0 > 0) .
eq output |= readers = false [owise] .

op writers  : -> Prop .
ceq run(store([L, o(objAttr f([t(t('data)), f([ 'wnum, L0 ] env)] oEnv)), I]
      [L0, int(I0), I0'] store) state) |= writers = true if (I0 > 0) .
eq output |= writers = false [owise] .

op program : -> Output .
…

red modelCheck(program, [] ( ~ ( readers /\ writers ) ) ) .
```

JavaPathFinder

- A VM-based model checker for Java
- Customizable via programming
  - Expressive but requires more effort
- http://javapathfinder.sourceforge.net/